KATHOLIEKE UNIVERSITEIT
LEUVEN

Faculty of Economics and Applied Economics

Proactive and reactive strategies fot resource-constrained project scheduling with uncertain resource availabilities

O. Lambrechts, Erik Demeulemeester and Willy Herroelen

DEPARTMENT OF DECISION SCIENCES AND INFORMATION MANAGEMENT (KBI)

KBI 0606

Proactive and Reactive Strategies for Resource-Constrained Project Scheduling with Uncertain Resource Availabilities

Olivier Lambrechts Erik Demeulemeester
Willy Herroelen
Department of Decision Sciences and Information Management
Research Center for Operations Management
Faculty of Economics and Applied Economics
Katholieke Universiteit Leuven (Belgium)

Abstract

Research concerning project planning under uncertainty has primarily focused on the stochastic resource-constrained project scheduling problem (stochastic RCPSP), an extension of the basic RCPSP, in which the assumption of deterministic activity durations is dropped. In this paper, we introduce a new variant of the RCPSP for which the uncertainty is modeled by means of resource availabilities that are subject to unforeseen breakdowns. Our objective is to build a robust schedule that meets the project due date and minimizes the schedule instability cost, defined as the expected weighted sum of the absolute deviations between the planned and actually realized activity starting times during project execution. We describe how stochastic resource breakdowns can be modeled, which reaction is recommended when a resource infeasibility occurs due to a breakdown and how one can protect the initial schedule from the adverse effects of potential breakdowns.

1 Introduction

Most of the research in project scheduling deals with the generation of an *initial project schedule* (*baseline schedule*) in a static and deterministic environment

with complete information. Traditional objective functions include minimizing the project makespan, leveling the resource usage over time, minimizing the total cost of acquiring the necessary resources, maximizing the project net present value and minimizing weighted earliness-tardiness penalty costs. For an extensive overview we refer to Brucker et al. (1999), Herroelen et al. (1998) and Demeulemeester & Herroelen (2002). In practice, however, these assumptions will hardly, if ever, be satisfied. As Aytug et al. (2005) indicate, it is often assumed that 'a system that works in a deterministic environment can be engineered to work under at least certain stochastic conditions'. Whereas for some problems this will indeed be the case (e.g. the WSPT rule minimizes the average weighted flow time for the single machine problem in the deterministic case and likewise the WSEPT rule minimizes the expected average weighted flow time in the stochastic case in the class of nonpreemptive static list policies and nonpreemptive dynamic policies (Pinedo 1995)), for others it will not. Therefore, we have to protect the initial baseline schedule from the adverse effects of possible disruptions. This protection is necessary because often project activities are subcontracted or executed by resources that are not exclusively reserved for the current project. A change in the starting times of such activities could lead to infeasibilities at the organizational level (in a multi-project context) or penalties in the form of higher subcontracting costs. A possible measure for the deviation between the initial schedule and the realized schedule is the *weighted instability cost*. It can be defined as the expected weighted absolute deviation between the planned and the actually realized activity starting times. The weight w_i, allocated to each activity i, reflects that activity's importance of starting it at its planned starting time in the initial schedule. More specifically, w_i denotes the marginal cost of deviating from the planned starting time of activity i during project execution. This marginal cost can be seen as an extra cost for having subcontractors start later than originally agreed or as an inventory cost for storing raw materials between the delivery time and the time they are needed. Minimizing instability then means that we are looking for the schedule that is least likely to get severely disrupted, i.e. a *robust schedule* that satisfies the precedence and resource constraints and does not exceed the due date set by the project's client. Meeting this due date during project execution is encouraged by giving a higher instability weight to the activity that signals the end of the project. Recent research (Leus (2003) and Van de Vonder et al. (2004)) considers this objective function for the case of project scheduling with stochastic activity durations. Other possible causes for uncertainty in project execution might be,

amongst others, inaccurate time estimates, bad weather conditions or unavailability of resources. In this paper we study the last of these possible causes. In machine scheduling, the problem of machines randomly breaking down has been reasonably well studied for the single machine (Mehta & Uzsoy 1999) and the job shop case (Mehta & Uzsoy 1998). However, except for Drezet (2005) we are not aware of any existing research in project scheduling dealing with the stochastic resource availability case. Drezet (2005) considers the problem of project planning with human resource constraints such as competences, a limit on the number of hours an employee works per day, vacation periods and unavailability of employees. A mathematical model as well as several algorithms are presented for building a robust schedule and for repairing a disrupted schedule.

2 Problem Statement

Aytug et al. (2005) stress the importance of taking potential disruptions into account when building and executing production schedules. The authors distinguish between predictive and reactive scheduling. Predictive (proactive) scheduling approaches try to accommodate uncertainties in advance whereas reactive approaches react after the fact.

Purely reactive project scheduling forgoes the construction of a baseline schedule and solely relies on the use of *scheduling policies* (Stork 2001) to decide on-line which activities are to be started at random decision points t that occur serially through time. These random decision points correspond with the completion times of the activities and the decision to start a precedence and resource feasible set of activities at time t can only be based on the information that has become available up to that time (non-anticipativity assumption).

Contrary to scheduling policies, *proactive scheduling* is based on the construction of a *baseline schedule*. This baseline schedule will guide schedule execution by providing for each activity its planned periods of execution as well as the resource units to be reserved during these execution periods. A baseline schedule is indispensable to coordinate resource allocation between multiple projects in a multi-project environment and to coordinate outsourced activities with subcontractors. The arguments Aytug et al. (2005) use to underline the importance of developing a production schedule in machine scheduling also apply to project scheduling. Some of the motivations they cite are: verifying the feasibility of executing the given tasks within a certain timeframe, providing

visibility of future actions for internal and external parties, offering degrees of freedom for reactive scheduling, evaluating performance and avoiding further problems.

Proactive scheduling relies on a baseline schedule that is made robust. A *robust baseline schedule* is a schedule that is not likely to get severely disrupted during schedule execution due to the occurrence of unforeseen events such as machine breakdowns.

The proactive baseline scheduling problem can now be formulated as follows:

minimize

$$\sum_{i \in N} w_i |E(\mathbf{s}_i) - s_i| \tag{2.1}$$

subject to

$$s_i + d_i \leqslant s_j \qquad \forall (i,j) \in A \tag{2.2}$$

$$\sum_{i : i \in S_t} r_{ik} \leqslant \mathbf{a}_k \qquad \forall t, \forall k \tag{2.3}$$

$$s_n \leqslant \delta_n \tag{2.4}$$

The objective of the problem is to minimize the weighted instability cost (2.1). The decision variables s_i represent the planned starting times for each activity i ($i \in N$ with $|N| = n$). Together, they define the baseline schedule which is represented by the vector $S = (s_1, s_2, \ldots, s_n)$. Because of the stochastic nature of the problem we cannot always stick to this baseline schedule. The real starting times are consequently stochastic variables that are represented by the stochastic vector $\mathbf{S} = (\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_n)$. We assume that a 'railroad-scheduling' approach is used. This means that activities are never started before their planned starting time ($\mathbf{s}_i \geqslant s_i$), implying that the objective function can be rewritten as $\sum_{i \in N} w_i (E(\mathbf{s}_i) - s_i)$. These planned starting times will have to respect certain constraints. The commonly used activity-on-node representation represents the project by means of a digraph $G = (N, A)$ where the set of nodes N represents the activities and the set of directed arcs A the finish-start, zero-lag precedence relations. When $(i,j) \in A$ we say that activity i is an immediate predecessor of activity j, implying that activity j may not start before activity i has finished. Precedence feasibility is enforced by constraints (2.2) where d_i is the deterministic duration of activity i. Constraints (2.3) enforce the renewable resource constraints. They imply that there does not exist a time period t and a

resource type k for which the cumulative resource requirements of the activities that are in progress during period t exceed the stochastic per-period availability $\mathbf{a}_k$ for the considered resource type. Here r_{ik} denotes the number of units of renewable resource type k required per period by activity i and S_t is the set of activities that are in progress at time period t. The last constraint (2.4) imposes the due date restriction.

Using the classification scheme of Herroelen et al. (2000), our problem can be classified as $m, 1, \mathbf{va}|cpm, \delta_n|\sum w_i(E(\mathbf{s}_i) - s_i)$. The first field specifies the resource characteristics: $(m, 1, \mathbf{va})$ refers to an arbitrary number of renewable resource types, each with a stochastic availability $\mathbf{a}_k$ that varies over time. The second field indicates the use of finish-start, zero-lag precedence relationships and a deterministic project due date. Finally, the last field shows the objective function, here the expected weighted instability cost.

The deterministic resource-constrained project scheduling problem is known to be strongly NP-hard. Allowing for stochastic resource availabilities complicates the problem. Moreover, the analytic evaluation of the objective function 2.1 is very cumbersome, so that one usually relies on simulation. For NP-hardness proofs of several cases of the scheduling problem for stability subject to a deadline and discrete disturbance scenario, we refer to Leus & Herroelen (2005).

Of course, it is still possible that the robust baseline schedule, despite the built-in protection, breaks during project execution (Davenport & Beck 2002). Therefore, we need a *reactive policy* that dictates how to revert to a feasible schedule that deviates as little as possible from the original baseline. *Proactive-reactive project scheduling* thus implies a combination of a proactive strategy for generating a protected baseline schedule S with a reactive strategy to resolve the schedule infeasibilities caused by the distortions that occur during schedule execution.

We introduce the example network in figure 1 to illustrate the various proactive and reactive strategies we present in this paper. This graph represents a project consisting of 10 activities. Above each activity node, we indicate its planned duration, its resource requirement of a single renewable resource type with a per period availability of 8 units and its instability weight. Note that activities 1 and 10 are dummy activities with a duration and a resource usage of 0. Activity 1 indicates the start of the project whereas activity 10 signals the end. The instability weight for activity 10 is much larger than the other instability weights in order to reflect the fact that in practice meeting the project due date

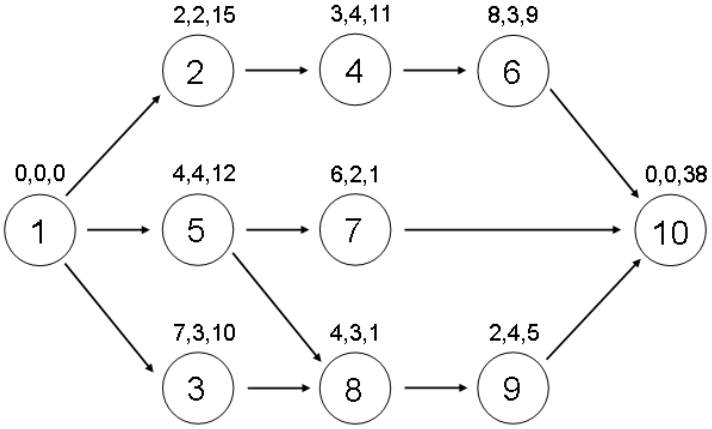

Figure 1: Example project network

is often deemed more important than meeting planned activity starting times. In this example we assume a project due date of 18. The baseline starting time of the dummy start activity is then set to the release date of the project (time period 0) whereas the dummy end activity is assumed to end at the project due date. Note that for ease of notation and illustration only one resource type is considered, but the examples as well as the algorithms presented in this paper are easily extensible to and will be tested for the multi-resource case.

In this paper we propose an approach for proactive scheduling in which three consecutive choices need to be made. Each choice consists of two options, giving us a total of 2^3 different strategies. First of all, we can either use the optimal solution for the RCPSP as a starting schedule or alternatively use a schedule in which activities that have a high impact on instability are scheduled as early as possible so that the probability that they get disrupted is lower. Secondly, one can decide to allow for resource slack or instead use the deterministic, maximum availabilities. Allowing for resource slack means that one plans the project considering a resource availability that is lower than the real availability so that a certain margin exists for absorbing resource breakdowns. Finally, it is either possible to protect individual activity starting times by inserting a time buffer of one or more time units in front of them or alternatively not to explicitly buffer activities at all.

The paper is structured as follows: in section 3, each of the proactive strate-

gies will be treated in more detail. The reactive policies are then presented in section 4, where we introduce two list scheduling policies and a tabu search procedure. Section 5 reports on extensive computational results obtained by testing the proactive-reactive procedures on a set of randomly generated test instances. Finally, section 6 presents our overall conclusions.

3 Proactive Strategies

3.1 Optimal solution for the RCPSP

The problem under study is an extension of the RCPSP. Since the objective of the deterministic RCPSP is to minimize the project makespan, the associated schedule will usually be very dense. This means that activities are scheduled compactly with as little resource and time slack as possible. In such a schedule even a minor disruption in the resource availabilities during a scheduling period will have a major impact on the starting times of all activities that are scheduled in subsequent periods. Therefore, it can be expected that such a schedule will perform very badly for the weighted instability cost objective. The optimal solution for the RCPSP associated with the project instance of figure 1 is given in figure 2. As expected, little free slack exists in this schedule. Free slack has been proposed by Al-Fawzan & Haouari (2005) as a metric for measuring the robustness of a schedule. It is defined as the amount of time an activity can be delayed beyond its planned starting time without forcing any other activities to be postponed. In our example the total free slack is equal to 6.

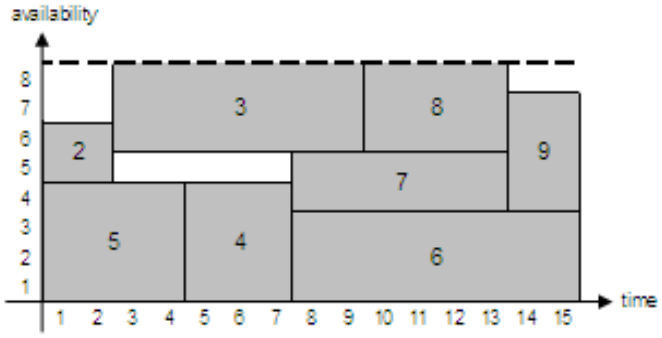

Figure 2: Minimal Makespan Schedule

3.2 Largest cumulative instability weight first

One way to improve schedule robustness is to schedule the activities in decreasing order of their cumulative instability weight. We define the *cumulative instability weight,* CIW_i, of activity i as $CIW_i = w_i + \sum_{j:j \in S_i^*} w_j$, with S_i^* the set of direct and indirect successors of activity i. Because disruptions propagate throughout the schedule, activities for which a change in starting time would have a high impact on instability are now less likely to get severely disrupted than activities with a lower impact since the former are scheduled earlier in time and are thus less prone to disruptions. The schedule is constructed in two phases. In the first phase a precedence feasible priority list is constructed with the activities in non-increasing order of their CIW_i (tie-breaker is lowest activity number). In the second phase, this priority list is transformed into a precedence and resource feasible schedule using the serial schedule generation scheme that was first introduced by Kelley (1963). The *serial schedule generation scheme* sequentially adds activities to the schedule until a feasible complete schedule is obtained. In each step, the next activity in the priority list is selected and for that activity the first precedence and resource feasible starting time is chosen. If we apply this heuristic to our example network, we obtain the vector of cumulative instability weights: $CIW = (102, 73, 54, 58, 57, 47, 39, 44, 43, 38)$. This vector corresponds to a priority list $L = (1, 2, 4, 5, 3, 6, 8, 9, 7, 10)$ that yields the schedule depicted in figure 3 when decoded using the serial schedule generation scheme. As expected, this schedule has a higher total free slack (13 compared to only 6 for the minimal makespan schedule).

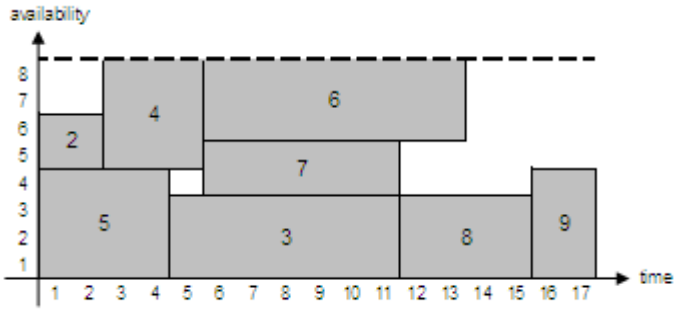

Figure 3: Largest CIW first schedule

3.3 Protection by means of resource slack

Baseline schedules can be protected against disruptions by including resource slack. This means that the project is planned using a resource availability (a_k^*) that is less than the maximum availability (a_k). In this case, a breakdown of one or more resource units will not always lead to a disruption of the schedule. This principle is inspired by the well-known result from factory physics that the lead time strongly increases in a non-linear fashion with increasing resource utilization and that therefore excess capacity is important (Hopp & Spearman 2001). The required size of the resource buffer will depend on the probability distribution of the resource availabilities. This probability distribution can be determined for the steady state if we assume that the time between two subsequent failures and the time until a broken resource unit is repaired are both exponentially distributed. This assumption can be motivated as follows: resources, whether they are humans, complex machinery or tools, can fail for a wide variety of reasons. We can therefore consider each resource unit to be composed of different components, each associated with a possible failure cause, with different times to failure. Let $N(t)$ be the total amount of breakdowns up to time t, split up by cause and component as follows: $N(t) = N_1(t) + N_2(t) + ... + N_m(t)$. If m is large enough and the times between counts for each breakdown cause are independent and identically distributed stochastic variables, then the resulting counting process $N(t)$ will follow a Poisson distribution (Hopp & Spearman 2001). Because a Poisson counting process corresponds to an exponential distribution of interarrival times (Girault 1959), the times between failures will be exponentially distributed. Unfortunately, this reasoning cannot be so easily applied to the times between repairs. However, it is analytically interesting but also practically acceptable to assume that these times are also exponentially distributed. In practice, it can be expected that there is high probability that repairs have a reasonably low duration but the possibility always exists that this duration strongly increases due to unexpected events such as extra complications, unavailability of spare parts, etc. Therefore, using the exponential distribution for modeling interrepair times seems a reasonable approximation.

The number of renewable resource units of type k actually available per period is a random variable $\mathbf{a}_k$. That means that each of the a_k resource units originally allocated to the project is subject to breakdowns characterized by a known mean time to failure ($MTTF_k$) and a mean time to repair ($MTTR_k$).

We can model the breakdowns as a birth-death process where the state j_k at any time is the number of inactive resource units of type k. A birth corresponds to a unit breaking down ($j_k = j_k + 1$) and a death to a unit having just been repaired ($j_k = j_k - 1$). Let $P_{ijk}(t)$ be the probability that j resource units of type k are inactive at time t given that i units were active at the starting period 0. In order to simplify the analysis of the breakdown process it would be easier if the probabilities we have to consider are independent of the starting conditions. This means that we will assume that the steady state has been reached, implying that the probabilities have converged to their steady state values or $\lim_{t\to\infty} P_{ijk}(t) = \pi_{jk}$. In order to simplify the notation we will omit the subscript k in the following paragraphs. An extension of the formulas to the multi-resource case is straightforward.

In order to calculate steady state probabilities, we need to know the birth and death rates in state j. We know that each resource unit breaks down at a rate λ with $\lambda = \frac{1}{MTTF}$ and that in state j, only $(a - j)$ out of the originally allocated a resource units are active. Therefore, the total rate at which breakdowns occur in state j is $\lambda_j = (a - j)\lambda$. Similarly, resource units are repaired at a rate μ with $\mu = \frac{1}{MTTR}$ and in state j, j units are inactive and can thus be repaired. The death rate in state j is then $\mu_j = j\mu$.

The steady state probabilities π_j for any birth-death process can be obtained by solving the system of flow balance equations (in the steady state the expected number of transfers into a state per time unit must equal the expected number of transfers out of that state per time unit). Consider the state j with $j \geqslant 1$ in figure 4.

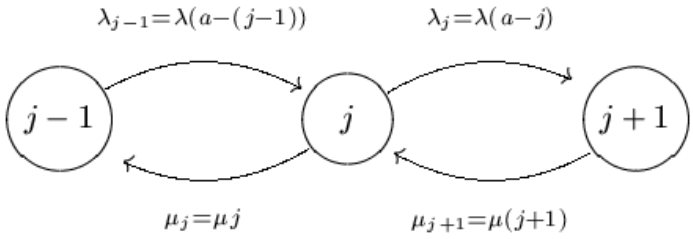

Figure 4: State Transition Diagram

If we write down the flow balance equation per state, we get the following

system of equations:

$$\begin{array}{lrcl}
(j=0) & \lambda_0\pi_0 & = & \mu_1\pi_1 \\
(j=1) & (\lambda_1+\mu_1)\pi_1 & = & \lambda_0\pi_0+\mu_2\pi_2 \\
\vdots & & & \\
j & (\lambda_j+\mu_j)\pi_j & = & \lambda_{j-1}\pi_{j-1}+\mu_{j+1}\pi_{j+1}
\end{array}$$

By expressing all the π_j's in terms of π_0 we get:

$$\begin{array}{rcl}
\pi_1 & = & \frac{\lambda_0\pi_0}{\mu_1} \\
\pi_2 & = & \frac{\lambda_0\lambda_1\pi_0}{\mu_1\mu_2} \\
\vdots & & \\
\pi_j & = & \frac{\lambda_0\lambda_1\ldots\lambda_{j-1}\pi_0}{\mu_1\mu_2\cdots\mu_j} \\
 & = & \frac{\lambda^j a(a-1)\ldots(a-j+1)\pi_0}{\mu^j j!} \\
 & = & \frac{\lambda^j a!\pi_0}{\mu^j j!(a-j)!} \\
 & = & \binom{a}{j}\frac{\lambda^j\pi_0}{\mu^j}
\end{array}$$

Because at any time we must be in some state, the sum of steady state probabilities $\sum_{j=0}^{a}\pi_j = 1$. Substituting the result for π_j in the previous equation yields:

$$\begin{array}{rcl}
\pi_0+\sum_{j=1}^{a}\frac{\lambda^j a!\pi_0}{\mu^j j!(a-j)!} & = & 1 \\
\pi_0 & = & \frac{1}{1+\sum_{j=1}^{a}\frac{\lambda^j a!}{\mu^j j!(a-j)!}}
\end{array}$$

Now, unless $\sum_{j=1}^{a}\frac{\lambda^j a!}{\mu^j j!(a-j)!}$ is infinite, π_0 and all the other steady state probabilities can be calculated. For our problem this condition will always be satisfied because a is finite and μ is assumed to be greater than 0.

Since we now know the discrete probability distribution of these availabilities, we can determine the expected value of the resource availability for resource type k as $E(\mathbf{a}_k) = \sum_{m=0}^{a_k} m\pi_m$ and use it as the buffered availability. In case this buffered availability is smaller than $\max_{i\in N} r_{ik}$, we increase the value until the activity with the highest resource demand for resource type k can be executed. The schedule is then built using the exact RCPSP method or the 'largest CIW

first'-method, but now with these adapted, buffered availabilities.

Note, however, that it is possible that the obtained resource buffered schedule exceeds the due date. Therefore, we need to add a mechanism that limits the maximal amount of resource buffering so that the due date constraint is not violated. If the resource buffered schedule turns out to be due date-infeasible, we determine the most constraining resource type, and progressively increase its availability up to the maximum (original) availability and re-execute the RCPSP or 'largest CIW first' procedure until the due date is met. We define the most constraining resource type as the resource type that leads to the highest decrease in schedule makespan when its buffered availability is increased by one unit. As a tie-breaker we choose the resource type with the smallest deviation between the expected resource availability and the adjusted buffered availability.

For our example, adding resource buffering to the minimal makespan schedule would yield the schedule depicted in figure 5. In this schedule, the original resource availability of eight units is reduced by one unit and the exact RCPSP procedure is executed for the project using this new, buffered availability, yielding the bottom schedule in figure 5.

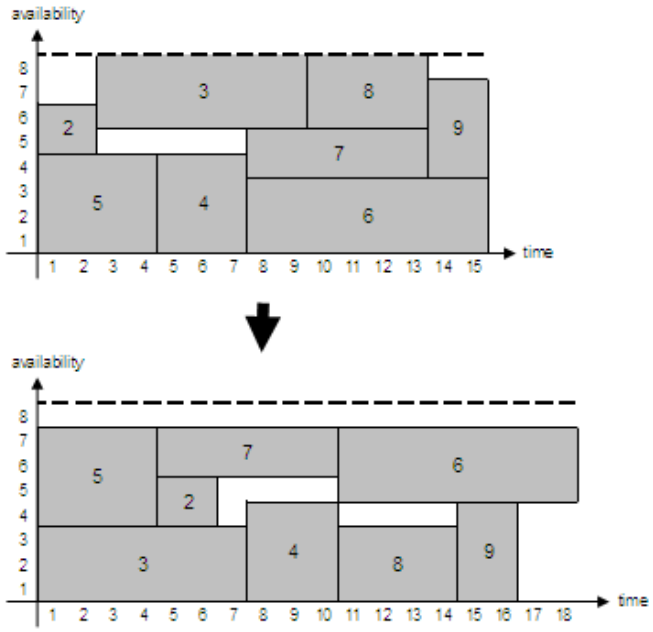

Figure 5: Resource buffering applied to a minimal makespan schedule

3.4 Time buffering

Instead of, or in addition to, resource buffering, another form of schedule protection can be used. Time buffering boils down to the inclusion of slack time in front of activities in order to absorb potential disruptions caused by earlier resource breakdowns and the resulting activity shifts. We start from a feasible baseline schedule to which protection is added by iteratively right-shifting activities with the aim of protecting the activity starting times as well as possible. Our objective is to insert a time buffer of size b_i in front of the starting times s_i of each activity i so that the expected instability is minimized while not exceeding the due date.

In order to set correct buffer sizes b_i for each activity i, we need to have a rough idea of the impact of the disruption (I_i) we can expect for that activity given a certain baseline schedule S. Disruptions, forcing the activity to start at a later point in time than originally planned, can occur amongst others when one or more of the activity's predecessors finish at a later time than expected because of resource breakdowns. First of all, we will describe how the expected duration increase (Δ_i) of an activity due to resource breakdowns can be approximated. We will then show how we can use these expected duration increases to determine which activities in the schedule will be affected. Finally, we will introduce a heuristic that selects the activities to be buffered and determines the proper buffer size, based on the expected impact of resource breakdowns on the activities constituting the project.

Imagine we have an activity i with a duration d_i and a single unit resource requirement with a per period availability of one unit, a mean time to failure equal to $MTTF$ and a mean time to repair equal to $MTTR$. We assume that after a broken down resource is repaired, execution can proceed from the point where it was interrupted. In this case the number of resource breakdowns activity i will experience is approximately equal to $\frac{d_i}{MTTF}$, leading to an expected duration increase of a magnitude $\frac{d_i}{MTTF} MTTR$.

However, this approach is very restrictive because first of all, we will usually work with resources for which the per period availability is higher than one. This means that when a resource unit breaks down, this will not always translate into a duration increase of the activity using that resource type. It is possible that there is sufficient resource slack to absorb the breakdown or that another activity is affected. For simplicity's sake let's however ignore these possibilities since they strongly complicate the calculations. In situations where the resource

units are not interchangeable this assumption is not even unrealistic. Secondly, activities will often have a resource requirement exceeding one unit. This again complicates our analysis because now, multiple resource units can be responsible for breakdowns and it is hard to analytically determine the aggregate effect. We choose to approximate the global duration extension by multiplying the expected duration increase with the resource usage of the considered resource: $\Delta_i = max_k(\frac{d_i}{MTTF_k} MTTR_k r_{ik})$. Even though this approach gives a reasonable estimate of the order of magnitude of the disruption, it is mathematically not correct and actually underestimates the real duration increase in the case of multiple resource units. However, its advantages are its great speed compared to calculating the real duration extension by means of simulation. Furthermore, our experimental results revealed that the use of simulation, rather than the simplified calculations, did not noticeably improve the objective function. For an overview of these results we refer the reader to the end of section 5.2.

The expected duration increases can now be used to approximate the impact of resource breakdowns on the given schedule S. For each non-dummy activity i we determine its direct and transitive predecessors $j \in P_i^*$. In case the predicted finish time of the considered predecessor $(s_j + d_j + \Delta_j)$ exceeds the planned starting time of activity i (s_i), it can be expected that there is a reasonable chance that activity i will be disrupted. The impact I_i on the objective function that can be attributed to the disruption of activity i caused by its predecessors j can be estimated as $I_i = \sum_{j \epsilon P_i^*} w_i(s_j + d_j + \Delta_j - s_i)$.

The non-dummy project activities are placed in a list Q in non-increasing impact order with the lowest activity number as tie-breaker. The first activity in list Q is selected and right-shifted with one time unit. Affected activities are likewise right-shifted with one time unit in order to keep the schedule precedence and resource feasible. In case the resulting schedule also respects the due date constraint, we can move to the next iteration by recalculating the expected impacts for each activity for the new schedule S' and building a new list Q'. In case the new schedule is not due date feasible we revert the move and select the next activity in Q; if no such activity can be found, the procedure is terminated. The pseudo-code for this approach is given in algorithm 1.

We illustrate the time buffering heuristic by describing some of the steps applied to our example network. The single renewable resource type in the example is assumed to have a mean time to failure equal to 18 and a mean time to repair equal to 5. Using the procedure to calculate the expected activ-

Algorithm 1 Time buffering heuristic

1: $\forall i : b_i = 0$
2: **for** $i = 2$ to $n - 1$ **do**
3: $\quad \Delta_i = max_k(\frac{d_i}{MTTF_k} MTTR_k r_{ik})$
4: **end for**
5: $\forall i : I_i = 0$
6: **for** $i = 2$ to $n - 1$ **do**
7: $\quad$ **for** $j \in P_i^*$ **do**
8: $\quad\quad$ **if** $s_j + d_j + \Delta_j > s_i$ **then**
9: $\quad\quad\quad I_i = I_i + w_i * (s_j + d_j + \Delta_j - s_i)$
10: $\quad\quad$ **end if**
11: $\quad$ **end for**
12: **end for**
13: $Q = N \setminus \{1, n\}$
14: sort Q in non-increasing order of I_i (tie-breaker is lowest activity number)
15: $b_{Q_{(1)}} = b_{Q_{(1)}} + 1$
16: determine S'
17: **if** $s'_n \leqslant \delta_n$ **then**
18: $\quad S = S'$
19: $\quad$ go to line 5
20: **else**
21: $\quad b_{Q_{(1)}} = b_{Q_{(1)}} - 1$
22: $\quad Q \setminus \{Q_{(1)}\}$
23: $\quad$ **if** $Q = \emptyset$ **then** exit **else** go to line 15
24: **end if**

ity duration increases detailed above, we obtain the duration extension vector $\Delta = (0, 1, 5, 3, 4, 6, 3, 3, 2, 0)$. We start from the largest CIW-baseline schedule shown as the top schedule in Figure 6. Calculating the expected impacts for this schedule yields the disruption impact vector $I = (0, 0, 0, 11, 0, 27, 3, 5, 20, 76)$. Activity 10 clearly has the highest impact value but is not considered for buffering because doing so would violate the due date of 18 since activity 10 is assumed to start and end at the project due date. Therefore activity 6 is selected and buffered with one time unit, yielding the second schedule in figure 6. The impact values are indicated between brackets. Activities that could not be selected for buffering because doing so would violate the due date are marked in white, whereas the activity that is selected for buffering is marked in black. Continuing the algorithm eventually yields the third schedule of figure 6, the corresponding vector of buffer sizes being $B = (0, 0, 0, 1, 0, 3, 2, 0, 1, 0)$.

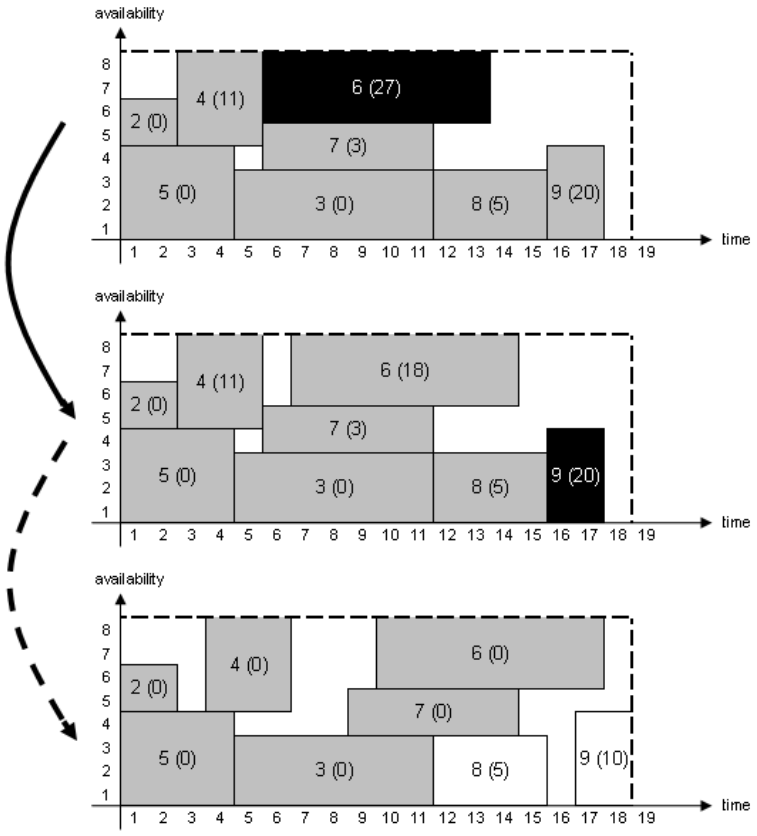

Figure 6: Time buffering applied to a 'maximum CIW first schedule'

4 Reactive Strategies

After the baseline schedule has been determined, project execution can start. However, no matter how much we try to protect the predictive schedule against possible disruptions, we can never totally eliminate their occurrence. The execution of the baseline schedule continues either until the completion of the dummy end activity, signaling the end of the project, or until a resource conflict is encountered. When a resource conflict occurs, this conflict will have to be resolved by postponing one or more activities in order to restore schedule feasibility. We assume that preemption is not allowed unless a resource infeasibility due to a resource breakdown is resolved by interrupting the execution of an activity that was in progress at the time of the breakdown. Furthermore, we assume that this interrupted activity then has to be restarted from scratch (*preempt-repeat*). An extension to a *preempt-resume* setting would be an interesting topic for further research.

4.1 List scheduling

A good reactive strategy restores schedule feasibility while minimizing the deviation from the baseline schedule and preventing future disruptions from occurring. A simple reactive strategy could rely on *list scheduling*.

A *random* precedence feasible priority list can serve as a benchmark. As an alternative, we rely on a *scheduled order list* that allows us to reschedule the activities in the order dictated by the schedule, while taking into account the new, reduced resource availabilities. More specifically, when a disruption occurs at time t^* we create a priority list L including the activities that are not yet completed, ordered in non-decreasing order of their baseline starting times.

The priority list is decoded into a feasible schedule using a *modified serial schedule generation scheme* and taking into account the known resource availabilities up to the current time period. The modification of the serial schedule generation scheme has to do with the case where the current activity taken from the list is in progress but not yet completed when the infeasibility occurs. This activity can be left unchanged, or it can be interrupted and repeated. The pseudocode for this procedure is given in algorithm 2. The new activity starting times are denoted as s'_i, the currently known availabilities as a'_{kt} and the set of direct predecessors of activity i as P_i.

Algorithm 2 Modified Serial Schedule Generation Scheme

given t^* = time period with resource infeasibility
given L = precedence feasible ordered list with activities $i : s_i + d_i \geqslant t^*$
for all k, t **do**
  if $t \leqslant t^*$ **then** $a'_{kt} = \mathbf{a}_{kt}$
  else $a'_{kt} = a_k$
end for
for $i = 1$ to n **do**
  if $i \notin L$ **then** $s'_i = s_i$
  else $s'_i = -1$
end for
for $p = 1$ to $|L|$ **do**
  if $s_{L_{(p)}} \leqslant t^*$ **then** $s'_{L_{(p)}} = s_{L_{(p)}}$
  else $s'_{L_{(p)}} = max(t^* + 1, max_{i \in P_{L_{(p)}}} s'_i + d_i)$
  while $\exists k, t : \sum_{i : i \in S_t} r_{ik} > a'_{kt}$ **do**
    $s'_{L_{(p)}} = t + 1$
  end while
end for

Activities selected from the list are scheduled as early as possible. For activities that are in execution during the time of disruption t^*, this means that the procedure first tries the current scheduled starting time. If this turns out to be infeasible, the procedure searches for feasibility by starting the activity in the next time period $(t^* + 1)$, and subsequent time periods if necessary. For activities that did not start yet, it is only necessary to consider the earliest precedence feasible starting time. Note that, as we stated in section 2, we never allow an activity to start before its baseline starting time.

4.2 Tabu search

Solutions may be improved by superimposing a tabu-search based improvement heuristic (Glover & Laguna 1993) on the priority list rule. This procedure will try to improve the starting solution by iteratively executing the best non-tabu precedence feasible adjacent interchange of two activities in the priority list. The objective is to find a precedence feasible ordering of activities that corresponds to a feasible schedule that deviates as little as possible from the baseline schedule. This deviation is measured by calculating the weighted sum of the absolute deviations between baseline and reactive schedule starting times. The advantage of tabu search is that by using a tabu list (a list of moves that

are forbidden for a number of iterations) the procedure can also choose non-improving moves so that it avoids getting stuck in local optima like traditional local search approaches. The procedure is explained in algorithm 3.

Algorithm 3 Tabu-search based reactive procedure

1: **set** $L^* = L$, $O^* = \sum_{i \in N} w_i |s'_i - s_i|$, $T = \sqrt{|L|}$
2: **while** (iter < MAXITER) **do**
3: $O_0 = 999999$
4: **for** $i = 2$ to $n - 2$ **do**
5: **if** $(L_{(i)}, L_{(i+1)}) \notin A$ **then**
6: exchange $L_{(i)}$ and $L_{(i+1)}$
7: perform adapted SSGS on L
8: $O = \sum_{i \in N} w_i |s'_i - s_i|$
9: **if** $s'_n \leqslant \delta_n$ and $O + freq_{L_{(i)}, s'_{L_{(i)}}} + freq_{L_{(i+1)}, s'_{L_{(i+1)}}} < O_0$ **then**
10: **if** $O_0 < O^*$ OR ($iter > tabu_{L_{(i)}, s'_{L_{(i)}}}$ AND $iter > tabu_{L_{(i+1)}, s'_{L_{(i+1)}}}$) **then**
11: store $i \rightarrow i*$
12: **end if**
13: **end if**
14: exchange $L_{(i)}$ and $L_{(i+1)}$
15: **end if**
16: **end for**
17: **if** $\exists i^*$ **then**
18: $freq_{L_{(i)}, s'_{L_{(i)}}} + 10$, $freq_{L_{(i+1)}, s'_{L_{(i+1)}}} + 10$
19: $tabu_{L_{(i)}, s'_{L_{(i)}}} = iter + T$, $tabu_{L_{(i+1)}, s'_{L_{(i+1)}}} = iter + T$
20: exchange $L_{(i^*)}$ and $L_{(i^*+1)}$
21: **if** $O = \sum_{i \in N} w_i |s'_i - s_i| < O^*$ **then**
22: $O^* = O$ AND $L^* = L$
23: **end if**
24: **end if**
25: **end while**
26: perform adapted SSGS on L^*

Our implementation considers a maximum number of iterations $MAXITER$ that has to be executed before the procedure ends and includes a frequency based penalty function to further prevent cycling. The tabu tenure is set to $\sqrt{|L|}$. The best solution that is found so far is stored in L^* and has an objective function value equal to O^*. O_0 then is the objective function value of the best adjacent

interchange found so far in the current iteration. The frequency based penalties are stored per pair (i, s_i) in the variables $freq_{i,s_i}$. Likewise, the tabu status is stored in the variables $tabu_{i,s_i}$.

In order to illustrate the reactive procedure we include the example in figure 7. The time buffered 'largest CIW first' schedule of figure 6 is disrupted in period 10 due to the breakdown of two resource units. Keeping the original schedule order, the partial priority list $(3, 7, 6, 8, 9, 10)$ is obtained, which yields the repaired schedule $S = (0, 0, 4, 3, 0, 10, 8, 11, 16, 18)$. However, improvement is possible. If we preempt and postpone activity 7 instead of activity 6 we obtain the partial priority list $(3, 6, 7, 8, 9, 10)$ and the schedule $S = (0, 0, 4, 3, 0, 9, 10, 11, 16, 18)$. The former schedule corresponds to a weighted schedule deviation cost of 9, whereas the latter schedule yields a weighted deviation cost of only 2.

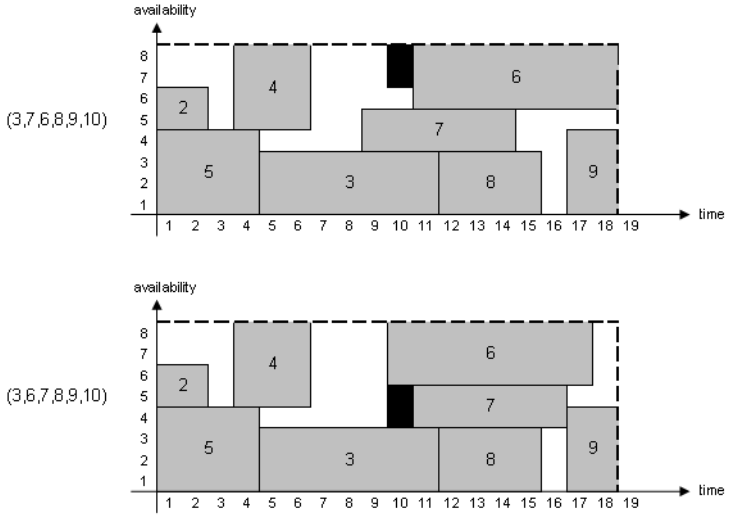

Figure 7: Reactive improvement procedure: original versus improved ordering

5 Results

5.1 Experiment

The above algorithms were coded in Microsoft Visual C++ 6.0 and executed on a Dell Optiplex GX270 workstation. In order to evaluate the instability objective we use simulation. For determining the optimal solution to the deterministic

RCPSP, we use the branch-and-bound algorithm developed by Demeulemeester and Herroelen (1992),(1997).

The instability weights w_i for all non-dummy activities are drawn from a discrete, triangularly shaped distribution between 1 and 10 with $P(\mathbf{w}_i = x) = 0.21 - 0.02x$. Corresponding to what can be expected in real-life projects, most activities will have a low instability weight whereas only a minority are more heavily penalized for being started later than planned. The instability weight of the dummy end activity represents the importance of meeting the projected due date and is set equal to β times the average of the instability weight distribution function, which is 3.85 for $P(\mathbf{w}_i = x)$. Because usually, meeting the project due date is deemed more important than starting each activity at the planned starting time, we set $\beta = 10$ for our experiment.

The project due date is derived from the minimal makespan schedule. In a static and deterministic environment, the lower bound on the makespan, (C_{max}^{RCPSP}), corresponds to the makespan of the schedule obtained when optimally solving the RCPSP. It seems reasonable to assume that the project manager will prefer a makespan that does not deviate too much from this lower bound. Therefore, we set the due date of the robust schedule at $C_{max}^{RCPSP}(1+\alpha)$, where the due date factor α is a parameter chosen by the project manager that constitutes the trade-off between project stability and project duration (Van de Vonder et al. 2005).

As mentioned in section 3.3, it can be shown that resource breakdowns can be modeled using exponential distributions with the parameters $MTTF_k$ and $MTTR_k$. We draw the $MTTR_k$ values from a uniform discrete distribution between 1 and 5. The values for $MTTF_k$ are drawn from a uniform discrete distribution between 50% and 150% of C_{max}^{RCPSP}.

As a test set for assessing the effectiveness of the proactive-reactive strategies, we use the 480 30-activity RCPSP instances of the well-known PSPLIB set of test problems (Kolisch & Sprecher 1997). Each combination of a proactive policy and a reactive policy was tested using 10 replications for each problem instance, each having different $MTTF_k$'s and $MTTR_k$'s. Furthermore, we used 50 iteration steps for the reactive tabu search.

5.2 Computational Results

The computational results are shown in tables 8, 9 and 10. The results shown in table 8 were obtained for a tight due date setting $\alpha = 15\%$, those in table 9

for $\alpha = 30\%$, and those in table 10 for an ample due date setting of $\alpha = 45\%$. We list the median values of the weighted instability costs over all projects and MTTF-MTTR scenarios for the 8 proactive scheduling combinations (time buffering or not, resource buffering or not, in combination with a minimum makespan schedule or a schedule obtained using 'largest CIW first') in combination with the three reactive procedures (random list scheduling, scheduled order list scheduling and tabu search). The numbers shown in italic in the last column give the average weighted instability cost value for each of the proactive scheduling rules, the italic numbers in the bottom row represent the average instability cost value for each of the reactive procedures.

Let us first have a look at the results for the proactive procedures. As could be expected, a proactive scheduling procedure using time buffering always seems to outperform procedures that do not. Of course, the possible improvement of time buffering directly depends on the degree of freedom offered to the time buffering algorithm for inserting buffers in the schedule. Therefore, whereas for the $\alpha = 30\%$ scenario, an average improvement of 40.05% can be observed for a minimal makespan schedule, this decreases to 33.44% for the 'largest CIW first'-schedule and only 22.92% for a resource buffered 'largest CIW first'-schedule. Similar results hold when varying the due date factor. For minimal makespan schedules, a 49.72% improvement seems to be possible when $\alpha = 45\%$ compared to only 23.01% when $\alpha = 15\%$.

Resource buffering performs quite well, offering average improvements of 60.53% for the minimal makespan case. Again, the improvement potential decreases as the due date factor is decreased. Taking into consideration the promising required computation time, resource buffering-based strategies cannot be neglected. In case of minimal makespan scheduling, for 84 of the 480 test instances the calculated buffered availability turned out to be too low. More specifically, this was the case for on average 25% of the considered MTTF-MTTR scenarios for each of these 84 problem instances. However, for 'largest CIW first' scheduling the picture is slightly different. For the networks for which 'largest CIW first' scheduling respected the project due date, the average availability turned out to be insufficient to obtain a feasible schedule for 20.28% of the cases when $\alpha = 15\%$, for 17.96% when $\alpha = 30\%$ and finally for 17.5% when $\alpha = 45\%$.

If resource buffering is not used, 'largest CIW first' usually performs better than minimal makespan scheduling. This is not surprising since we actively try to improve the objective function value of the minimal makespan sched-

ule. However, for $\alpha = 30\%$ the combination of minimal makespan scheduling, resource buffering and time buffering actually seemed to outperform the combination of 'largest CIW first', resource buffering and time buffering. A possible reason might be that minimal makespan scheduling creates a shorter schedule in which more opportunities for time buffering exist. On the other hand, similar results were not found for $\alpha = 45\%$. For $\alpha = 45\%$ the combination based on 'largest CIW first' performed slightly better than the one based on minimal makespan but the difference is so small it is almost negligible. Also note that for $\alpha = 15\%, 30\%, 45\%$, determining the 'largest CIW first'-schedule turned out to be impossible in respectively $28.12\%, 3.75\%$ and 0.00% of the instances, because the corresponding schedule exceeded the due date.

The results for the reactive strategies are as expected. Random list scheduling performs worst. Scheduled order list scheduling offers a significant performance increase. Tabu search allows further improvements upon scheduled order list scheduling.

α = 15 %			random list	scheduled order	tabu search	
no time buf	no res buf	min Cmax	1370,00	348,00	302,50	*673,50*
		largest CIW first	1025,40	329,65	261,05	*538,70*
	res buf	min Cmax	633,55	121,55	110,85	*288,65*
		largest CIW first	521,20	104,70	85,40	*237,10*
time buf	no res buf	min Cmax	1117,90	233,70	203,95	*518,52*
		largest CIW first	899,00	136,80	116,20	*384,00*
	res buf	min Cmax	546,85	101,40	89,70	*245,98*
		largest CIW first	482,70	72,40	69,00	*208,03*
			824,58	*181,03*	*154,83*	

Figure 8: Median of weighted instability for alpha = 15 %

α = 30 %			random list	scheduled order	tabu search	
no time buf	no res buf	min Cmax	1099,10	297,95	259,10	*552,05*
		largest CIW first	1026,75	257,50	205,25	*496,50*
	res buf	min Cmax	470,40	95,85	81,10	*215,78*
		largest CIW first	410,80	77,05	63,05	*183,63*
time buf	no res buf	min Cmax	788,85	105,90	98,20	*330,98*
		largest CIW first	795,10	107,20	89,10	*330,47*
	res buf	min Cmax	314,30	44,35	37,25	*131,97*
		largest CIW first	327,35	54,60	42,70	*141,55*
			654,08	*130,05*	*109,47*	

Figure 9: Median of weighted instability for alpha = 30 %

Table 11 lists the average required CPU times in seconds. Proactive policies based on 'largest CIW first' are computationally very cheap. This is not

α = 45 %			random list	scheduled order	tabu search	
no time buf	no res buf	min Cmax	991,35	287,10	256,60	*511,68*
		largest CIW first	898,35	235,20	191,80	*441,78*
	res buf	min Cmax	383,35	95,50	80,65	*186,50*
		largest CIW first	339,55	65,45	50,85	*151,95*
time buf	no res buf	min Cmax	631,20	76,00	64,65	*257,28*
		largest CIW first	610,15	69,30	59,90	*246,45*
	res buf	min Cmax	202,40	24,90	22,40	*83,23*
		largest CIW first	200,90	24,50	21,35	*82,25*
			532,16	*109,74*	*93,53*	

Figure 10: Median of weighted instability for alpha = 45 %

				α = 15 %	α = 30 %	α = 45 %
proactive	no time buf	no res buf	min Cmax	0,066	0,066	0,067
			largest CIW first	0,000	0,000	0,000
		res buf	min Cmax	0,149	0,075	0,076
			largest CIW first	0,001	0,001	0,000
	time buf	no res buf	min Cmax	0,090	0,084	0,079
			largest CIW first	0,029	0,020	0,015
		res buf	min Cmax	0,170	0,092	0,088
			largest CIW first	0,026	0,020	0,015
reactive	random list			0,000	0,000	0,000
	scheduled order			0,000	0,000	0,000
	tabu search			0,044	0,044	0,044

Figure 11: Average CPU time in seconds

surprising given the simple schedule construction procedures based on the serial schedule generation scheme. Minimal makespan scheduling is slower because of the exact branch-and-bound approach. Especially when looking at the case of $\alpha = 15\%$ we see that the minimal makespan procedure combined with resource buffering is rather slow. This is probably due to the fact that given the restrictive due date-factor the procedure has to be executed a number of times until the buffered availability allows for the creation of a due date-feasible schedule. The time buffering procedure is likewise very fast with decreasing computation times as the due date-factor increases. The simple reactive policies are very fast, tabu search, however, is computationally slightly more expensive.

Finally, in order to evaluate the impact of using the simplifying calculation of section 3.4 for computing the activity duration extensions Δ_i, we performed a small computational experiment in which this simplifying calculation is compared with simulation. Table 12 shows the results of this computational experiment using a subset of the instances used in the experiment detailed in section 5.1 for a due date factor of 30%. The time buffering approach based on simulation is compared with the approach based on approximation for each of the two

other proactive options (minimal makespan or 'largest CIW first' and resource buffering or not) combined with each of the three reactive strategies. We used a total of 1000 runs per activity to simulate the expected duration increase. We can conclude from these results that calculating Δ_i using simulation usually yields a better performance for the minimal makespan startschedule but the gain is negligible when compared with the additional computational effort (CPU times increased on average with a factor of 55). Furthermore, for a 'largest CIW first' startschedule simulation even seems to perform worse in most cases.

			random list	scheduled order	tabu search	
simulation	no res buf	min Cmax	1128,99	302,54	273,73	*568,42*
		largest CIW first	995,97	336,33	291,90	*541,40*
	res buf	min Cmax	518,01	221,43	211,27	*316,90*
		largest CIW first	581,83	260,14	233,57	*358,52*
approximation	no res buf	min Cmax	1166,17	326,78	291,31	*594,75*
		largest CIW first	968,59	323,27	284,89	*525,58*
	res buf	min Cmax	537,41	232,03	220,83	*330,09*
		largest CIW first	637,48	250,05	228,60	*372,04*

Figure 12: Comparison of approximation with simulation

6 Conclusion

In this paper we gave an overview of the challenges a project manager has to deal with in an environment characterized by uncertain resource availabilities. We gave an overview of the literature on scheduling under uncertainty and underlined the necessity of building a proactive baseline schedule for minimizing weighted instability. For the generation of a robust baseline schedule we proposed eight strategies. First of all, a starting schedule can be built using for example minimal makespan scheduling or largest CIW scheduling. This schedule can then be protected against the effects of disruptions by using the average resource availabilities, obtained from steady-state probability calculations, instead of the given deterministic availabilities. Alternatively or additionally, protection can be added in the form of explicitly inserted idle time. To determine where and in what amount to insert these so called time buffers, we developed a time buffer allocation heuristic based on the estimation of the expected impacts of activity duration prolongations due to resource breakdowns. The advantages of 'largest CIW first'-scheduling combined with resource buffering and time buffering immediately become apparent when comparing the weighted

instability results with those from a minimal makespan strategy without buffering. From a simulation experiment using the PSPLIB set of test problems we were able to observe an average improvement of 84%.

Unfortunately, no matter how much one tries to protect the initial schedule, the occurrence of disruptions during project execution can never be totally avoided. Therefore reactive policies, indicating how to restore schedule feasibility after the occurrence of a resource breakdown, are required. We proposed three reactive policies to resolve infeasibilities resulting from schedule disruptions. A random order rule was included for benchmarking purposes, the scheduled order rule seems to perform reasonably well in comparison but can still be improved by tabu search at the cost of an increase in computation time. Here the computational experiment also immediately showed the advantages of using an intelligent reactive strategy. Using scheduled order instead of random order allowed us to obtain results that were on average 79% better. A further improvement of about 15% was possible when superimposing a tabu-search improvement procedure on the original order rule.

In conclusion, we were able to show that a combination of largest CIW scheduling, resource buffering, time buffering and a reactive strategy based on an improvement of the scheduled order rule, allows for an improvement of the objective function of 96% compared to minimal makespan scheduling without any buffering using a dumb random order strategy.

References

Al-Fawzan, M. A. & Haouari, M. (2005). A bi-objective model for robust resource-constrained project scheduling. *International Journal of Production Economics*, 96, pp 175–187.

Aytug, H., Lawley, M., McKay, K., Mohan, S. & Uzoy, R. (2005). Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research*, 161, pp 86–110.

Brucker, P., Drexl, A., Möhring, R., Neumann, K. & Pesch, E. (1999). Resource-constrained project scheduling: Notation, classification, models and methods. *European Journal of Operational Research*, 112, pp 3–41.

Davenport, A. & Beck, J. (2002). A survey of techniques for scheduling with uncertainty. *Unpublished manuscript.*

Demeulemeester, E. & Herroelen, W. (1992). A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. *Management Science*, 38, pp 1803–1818.

Demeulemeester, E. & Herroelen, W. (1997). New benchmark results for the resource-constrained project scheduling problem. *Management Science*, 43, pp 1485–1492.

Demeulemeester, E. & Herroelen, W. (2002). *Project scheduling - A research handbook*. Vol. 49 of *International Series in Operations Research & Management Science*. Kluwer Academic Publishers, Boston.

Drezet, L.-E. (2005). *Résolution d'un problème de gestion de projets sous contraintes de ressources humaines: De l'approche prédictive à l'approche réactive*. PhD thesis. Université François Rabelais Tours, France.

Girault, M. (1959). *Initiation aux Processus Aléatoires*. Dunod, Paris.

Glover, F. & Laguna, M. (1993). *Tabu Search*. Blackwell Scientific, Oxford. pp 70–141. In C. Reeves (Editor): Modern Heuristic Techniques for Combinatorial Problems.

Herroelen, W., De Reyck, B. & Demeulemeester, E. (1998). Resource-constrained scheduling: A survey of recent developments. *Computers and Operations Research*, 25, pp 279–302.

Herroelen, W., De Reyck, B. & Demeulemeester, E. (2000). On the paper "Resource-constrained project scheduling: Notation, classification, models and methods" by Brucker et al.. *European Journal of Operations Research*, 128(3), pp 221–230.

Hopp, W. & Spearman, M. (2001). *Factory physics : Foundations of manufacturing management*. McGraw-Hill.

Kolisch, R. & Sprecher, A. (1997). PSPLIB - A project scheduling library. *European Journal of Operational Research*, 96, pp 205–216.

Leus, R. (2003). *The generation of stable project plans*. PhD thesis. Department of applied economics, Katholieke Universiteit Leuven, Belgium.

Leus, R. & Herroelen, W. (2005). The complexity of machine scheduling for stability with a single disrupted job. *Operations Research Letters*, 33, pp 151–156.

Mehta, S. & Uzsoy, R. (1998). Predictive scheduling of a job shop subject to breakdowns. *IEEE Transactions on Robotics and Automation*, 14, pp 365–378.

Mehta, S. & Uzsoy, R. (1999). Predictable scheduling of a single machine subject to breakdowns. *Int. J. Computer Integrated Manufacturing*, 12, pp 15–38.

Pinedo, M. (1995). *Scheduling - Theory, Algorithms and Systems*. Prentice Hall, Englewood Cliffs, New Jersey.

Stork, F. (2001). *Stochastic Resource-Constrained Project Scheduling*. PhD thesis. Technical University of Berlin, School of Mathematics and Natural Sciences.

Van de Vonder, S., Demeulemeester, E., Herroelen, W. & Leus, R. (2004). The trade-off between stability and makespan in resource-constrained project scheduling. *Research Report 0423*. Department of applied economics, Katholieke Universiteit Leuven, Belgium.

Van de Vonder, S., Demeulemeester, E., Herroelen, W. & Leus, R. (2005). The use of buffers in project management: The trade-off between stability and makespan. *International Journal of Production Economics*, 97, pp 227–240.